(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
13 February 2003 (13.02.2003)

PCT

(10) International Publication Number
WO 03/013102 A1

(51) International Patent Classification[7]: H04L 29/06, 29/14

(21) International Application Number: PCT/IB01/01382

(22) International Filing Date: 2 August 2001 (02.08.2001)

(25) Filing Language: English

(26) Publication Language: English

(71) Applicant *(for all designated States except US)*: **SUN MICROSYSTEMS, INC.** [US/US]; 901 San Antonio Road, Palo Alto, CA 94303 (US).

(72) Inventor; and

(75) Inventor/Applicant *(for US only)*: **REISS, Christophe** [FR/FR]; 71, rue Abbé Grégoire, F-38000 Grenoble (FR).

(74) Agent: **PLACAIS, Jean-Yves**; Cabinet Netter, 40, rue Vignon, F-75009 Paris (FR).

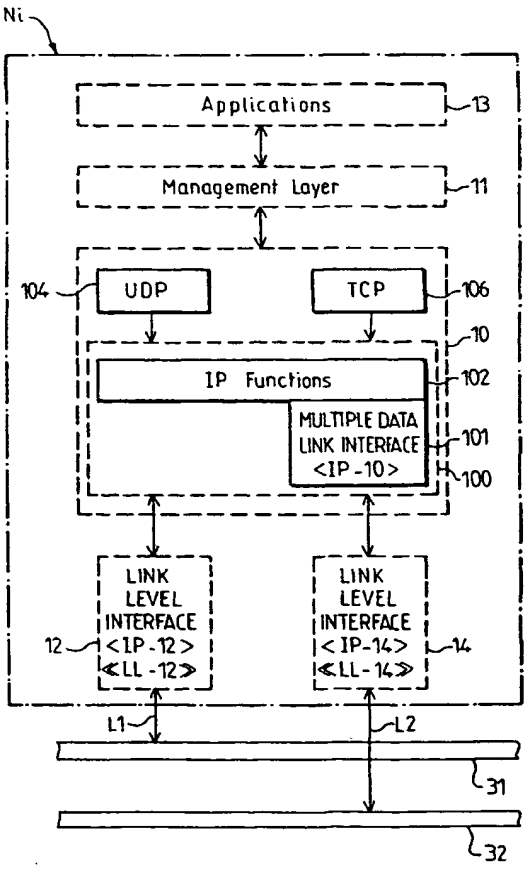
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*

[Continued on next page]

(54) Title: FILTERING REDUNDANT PACKETS IN COMPUTER NETWORK EQUIPMENTS



(57) Abstract: A node NI has a protocol stack (10), between application (13) and management (11) layers, and two or more Link level interfaces (12, 14) to network links (31, 32). Protocol stack (10) has a multiple data link interface (101), which may duplicate packets at transmission, to be sent through links (31, 32), respecitvely. A memory area is reserved. At reception, an idnetifier (X) is calculated for each incoming packet e.g. form its IP header, and is searched in at least a portion of the reserved memory area; if X is not found (or not found for the source node), the packet identifier (X) is stored in a currently reserved portion of the memory area.

WO 03/013102 A1

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

Filtering redundant packets in computer network equipments

This invention relates to network equipments, as used for example in telecommunication systems.

In such equipments, computer stations or nodes are interconnected through a network medium or link. The link may have to be at least partially duplicated to meet reliability constraints. This is called link redundancy. It is now assumed by way of example that data are exchanged between the nodes in the form of packets. Considering a given packet sent from a source node to a destination node, redundancy means that two or more copies of that packet are sent to the destination node through two or more different networks, respectively. The copies of the packet will usually reach the destination node at different times. Thus, a first one of the packet copies is normally processed in the destination node; when arriving, the other copy or copies ("redundant packets") will be processed in a manner which may depend e.g. upon the transport protocol and/or the user application.

The known Transmission Control Protocol (TCP) has a built-in capability to suppress redundant packets. However, this built-in capability involves potentially long and unpredictable delays. On another hand, the known User Datagram Protocol (UDP) has no such capability; in this case, suppressing redundant packets is a task for user applications.

A general aim of the present invention is to provide advances with respect to such mechanisms.

In a first aspect, this invention offers redundant packet filtering software code, comprising code for defining:

- a memory manager, having a reserved memory area, in which it defines a currently designated portion of memory, and
- an incoming packet manager, responsive to an incoming packet for:
    * determining an identifier (X) from information contained in the incoming packet,
    * searching the identifier (X) in at least a portion of the reserved memory area,
    * if a first condition related to the search is met, storing the identifier (X) in the currently designated portion of memory, and

* if a second condition related to the search is met, filtering out the incoming packet as redundant.

In a second aspect, this invention offers a method of processing redundant packets comprising the steps of:

a. determining an identifier (X) from information contained in an incoming packet (630),
b. searching the identifier (X)in at least a portion of a reserved memory area (560), and
c. processing the packet, the processing comprising:
   c1. if a first condition related to the search is met, storing the identifier (X) in a currently designated portion of the reserved memory area, and
   c2. if a second condition related to the search is met, filtering out the incoming packet as redundant.

In both cases, as it will be seen:
- the identifier may, or may not, take fragment-containing packets into consideration;
- the identifier may, or may not, take a packet source identifier into consideration, depending upon whether the memory is organized with a particular area for each source address, or not;
- the first condition may include the fact the packet identifier is not found in the memory, at least for un-fragmented packets. Other alternatives of the first and second conditions will be described hereinafter.

This invention also covers a node, having a protocol stack, and a multiple data link interface, capable of at least partially implementing the above functions, and/or their developments to be described hereinafter.

This invention further covers a cluster having a plurality of nodes interconnected through a redundant link, in which at least two of the nodes are arranged for implementing the above functions, and/or their developments to be described hereinafter.

Other alternative features and advantages of the invention will appear in the detailed description below and in the appended drawings, in which :

- figure 1 is a general diagram of a telecommunication network system, as an exemplary context in which this invention may be applicable;

- figure 2 shows a group of stations or nodes interconnected through two different links;

- figure 3 is an exemplary block diagram of a computer station or node, incorporating an exemplary embodiment of this invention;

- figure 4 shows an exemplary format of an IP header in a packet;

- figure 5 shows a flow chart of a packet transmission in redundant mode;

- figure 6 shows the initial reception of redundant packets;

- figure 7 shows the structure of an exemplary software module used in this invention;

- figure 8 is a general flow-chart of the discrimination of received packets;

- figure 9 shows an exemplary detailed embodiment of operation 640 in figure 8;

- figure 10 is an exemplary memory arrangement, for implementing this invention;

- figure 11 shows an exemplary detailed embodiment of operation 650 in figure 8; and

- figure 12 shows an exemplary detailed embodiment of operation 670 in figure 8.

As they may be cited in this specification, Sun, Sun Microsystems, Solaris, ChorusOS are trademarks of Sun Microsystems, Inc. SPARC is a trademark of SPARC International, Inc.

Additionally, the detailed description is supplemented with the following Exhibits:
- Exhibit A contains formulas used in this specification.

In the foregoing description, references to the Exhibits may be made directly by the Exhibit or Exhibit section identifier. One or more Exhibits are placed apart for the purpose of clarifying the detailed description, and of enabling easier reference. They nevertheless form an integral part of the description of the present invention. This applies to the drawings as well.

Now, making reference to software entities imposes certain conventions in notation. For example, in the detailed description, *Italics* (and/or the quote sign ") may be used when deemed necessary for clarity.

Figure 1 illustrates an exemplary simplified telecommunication network system. Terminal devices (TD) like 1 are in charge of transmitting data, e.g. connection request data, to base transmission stations (BTS) like 3. A such base transmission station 3 gives access to a communication network, under control of a base station controller (BSC) 4. The base station controller 4 comprises communication nodes, supporting communication services ("applications"). Base station controller 4 also uses a mobile switching center 8 (MSC), adapted to orientate data to a desired communication service (or node), and further service nodes 9 (General Packet Radio Service, GPRS), giving access to network services, e.g. Web servers 19, application servers 29, data base server 39. Base station controller 4 is managed by an operation management center 6 (OMC).

Certain items in the system of figure 1, e.g. the base station controllers 4, may comprise one or more groups of nodes, or clusters, exchanging data through two or more redundant networks. Reference to base station controllers is purely exemplary, since other components in a telecommunication systems, as well as in other types of data or message exchanging systems, may have a similar organization.

Figure 2 now shows a cluster having C nodes N1, N2, .. N_C, interconnected through two different links 31 and 32. In the foregoing description, Ni and Nj will designate two nodes, with i and j being comprised between 1 and C, inclusively. The network links or channels 31, 32 as used may be high speed network channels with equivalent bandwidth and latency. However,

other channels may be also used, e.g. heterogeneous networks. In a purely exemplary embodiment, links 31 and 32 are arranged as Ethernet physical networks.

Figure 3 shows an exemplary node Ni, in which the invention may be applied. Node Ni comprises, from top to bottom, applications 13, management layer 11, network protocol stack 10, and link level interfaces 12 and 14, respectively interacting with network links 31 and 32 (also shown in figure 2). Node Ni may be part of a local or global network; in the foregoing exemplary description, the network is Internet, by way of example only. It is assumed that each node may be uniquely defined by a portion of its Internet address. Accordingly, as used hereinafter, "Internet address" or "IP address" means an address uniquely designating a node in the network being considered (e.g. a cluster), whichever network protocol is being used. Although Internet is presently convenient, no restriction to Internet is intended.

Thus, in the example, network protocol stack 10 comprises:

- an Internet interface 100, having conventional Internet protocol (IP) functions 102, and a multiple data link interface 101,
- above Internet interface 100, message protocol processing functions, e.g. an UDP function 104 and/or a TCP function 106.

Network protocol stack 10 is interconnected with the physical networks through first and second link level interfaces 12 and 14, respectively. These are in turn connected to first and second network channels 31 and 32, via couplings L1 and L2, respectively. More than two channels may be provided, enabling to work on more than two copies of a packet.

Link level interface 12 has an Internet address <*IP_12*> and a Link level address <<*LL_12*>>. Incidentally, the doubled triangular brackets (<< ... >>) are used only to distinguish link level addresses from Internet addresses. Similarly, Link level interface 14 has an an Internet address <*IP_14*> and a Link level address <<*LL_14*>>. In a specific embodiment, where the physical network is Ethernet-based, interfaces 12 and 14 are Ethernet interfaces, and <<*LL_12*>> and <<*LL_14*>> are Ethernet addresses.

IP functions 102 comprise encapsulating a message coming from upper layers 104 or 106 into a suitable IP packet format, and, conversely, de-encapsulating a received packet before delivering the message it contains to upper layer 104 or 106.

In redundant operation, the interconnection between IP layer 102 and link level interfaces 12 and 14 occurs through multiple data link interface 101. The multiple data link interface 101 also has an IP address *<IP_10>*, which is the node address in a packet sent from source node Ni.

References to Internet and Ethernet are exemplary, and other protocols may be used as well, both in stack 10, including multiple data link interface 101, and/or in link level interfaces 12 and 14.

Furthermore, where no redundancy is required, IP layer 102 may directly exchange messages with anyone of interfaces 12,14, thus by-passing multiple data link interface 101.

Now, when circulating on any of links 31 and 32, a packet may have several layers of headers in its frame: for example, a packet may have, encapsulated within each other, a transport protocol header, an IP header, and a link level header.

As shown in figure 4, an exemplary IP header may comprise the following fields:

- a destination IP address 220;
- a source IP address 221;
- a header checksum 222;
- a Time To Live (TTL)223;
- a protocol identifier (*IP-PROT*) 224;
- a zone 225 containing fragmentation flags, and fragment offsets (*IP-OFF*);
- an IP identification (*IP-ID*) 226;
- an IP total length 227;
- a type of service (T.O.S.) 228;
- a Header Length (Internet Header Length) 229; and
- a version identifier 230.

Certain of these fields are defined at the level of network protocol stack 10. For a packet corresponding to a complete data message, fields 220, 221, 224 and 226 are sufficient to identify the data message. Optionally, a data message may be split into a plurality of fragments, sent through different packets. In this case, a packet corresponding to a fragment of a data message will have its field 225 completed with an indication of the position of the fragment in the data message.

The other fields are mere service fields: for example, field 223 (TTL) determines the time after which the packet may be destructed.

In this specification, "packet header" refers to information attached to a packet, and indicating e.g. the source, the destination, and other service information. No restriction is intended to the packet header being at the beginning of the packet: it may be at the end as well, or even dispersed within the message data.

The operation of sending a packet Ps in redundant mode will now be described with reference to figure 5.

At 500, network protocol stack 10 of node Ni receives a packet Ps from application layer 13 through management layer 11. At 502, packet Ps is encapsulated with an IP header, in which:
- field 220 comprises the address of a destination node, which is e.g. the IP address IP_10(j) of the destination node Nj in the cluster;
- field 221 comprises the address of the source node, which is e.g. the IP address IP_10(i) of the current node Ni.

Both addresses IP_10(i) and IP_10(j) may be "intra-cluster" addresses, defined within the local cluster, e.g. restricted to the portion of a full address which is sufficient to uniquely identify each node in the cluster.

In protocol stack 10, multiple data link interface 101 has data enabling to define two or more different link paths for the packet (operation 504). Such data may comprise e.g.:

- a routing table, which contains information enabling to reach IP address IP_10(j) using two different routes (or more) to Nj, going respectively through distant interfaces IP_12(j) and IP_14(j) of node Nj;
- link level decision mechanisms, which decide the way these routes pass through local interfaces IP_12(i) and IP_14(i), respectively;
- additionally, an address resolution protocol (e.g. the ARP of Ethernet) may be used to make the correspondence between the IP address of a link level interface and its link level (e.g. Ethernet) address.

At this time, packet Ps is duplicated into two copies Ps1, Ps2 (or more, if more than two links 31, 32 are being used). In fact, the copies Ps1, Ps2 of packet Ps may be elaborated within network protocol stack 10, either from the beginning (IP header encapsulation), or at the time the packet copies will need to have different encapsulation, or in between.

At 506, each copy Ps1, Ps2 of packet Ps now receives a respective link level header or link level encapsulation. Each copy of the packet is sent to a respective one of interfaces 12 and 14 of node Ni, as determined e.g. by the above mentioned address resolution protocol.

In a more detailed exemplary embodiment, multiple data link interface 101 in protocol stack 10 may prepare (at 511) a first packet copy Ps1, having the link level destination address LL_12(j), and send it through e.g. interface 12, having the link level source address LL_12(i). Similarly, at 512, another packet copy Ps2 is provided with a link level header containing the link level destination address LL_14(j), and sent through e.g. interface 14, having the link level source address LL_14(i).

On the reception side, several copies of a packet, now denoted generically Pa should be received from the network in node Nj. The first arriving copy is denoted Pa1; the other copy or copies are denoted Pa2, and also termed "redundant" packet(s), to reflect the fact they bring no new information.

As shown in figure 6, one copy Pa1 should arrive through e.g. Link level interface 12-j, which, at 601, will de-encapsulate the packet, thereby removing the link level header (and address), and pass it to protocol stack 10(j) at 610. One additional copy Pa2 should also arrive through Link

level interface 14-j which will de-encapsulate the packet at 602, thereby removing the link level header (and address), and pass it also to protocol stack 10(j) at 610.

Thus, protocol stack 610 normally receives two identical copies of the IP packet Pa, within the flow of other packets. This invention will enable (i) discriminating between a first incoming packet Pa1 and one or more redundant following packets Pa2, and (ii) filtering the packet data. The filtering will depend upon the fact a message is fragmented between several packets or not, if such a fragmentation is authorized. Since the ultimate purpose is in most cases filtering, the word "filtering", as used here, may encompass both discriminating and filtering. It should however be kept in mind that "discriminating" is the basic function.

It should now be recalled that, amongst various transport internet protocols, the messages may use the Transmission Control Protocol (TCP), when passing through function or layer 106; TCP has its own capability to suppress redundant packets but with long and unpredictable delays. The messages may also use the User Datagram Protocol (UDP), when passing through function or layer 104; the User Datagram Protocol relies on application's capability to suppress redundant packets, in the case of redundancy, which is also long and resource consuming.

Incoming packet copies have an IP header according to figure 4. The transport protocol (TCP, UDP, or others) being used for a packet is specified in field 224 of the IP header (or, alternatively, in a separate transport protocol header).

This invention may be viewed as providing, at reception side, a filtering function which operates independently of the transport internet protocol being used, i.e whether e.g. TCP or UDP in the case of Internet, or another protocol. This invention is also compatible with the existing transport protocols: the built-in TCP processing of redundant packets may be kept in function; in case of UDP, the processing of redundant packets by user applications may also be kept in function.

Thus, network protocol stack 10 comprises a filtering function to detect and reject redundant packets. The filtering function may be located in multi data link interface 101, or in IP layer 102, or in a distinct function module.

In accordance with an aspect of this invention, information contained in the IP headers of packets may be used for discriminating packets when they arrive to network protocol stack 10. To this effect, this information is used to build distinctive identifiers or "footprints" of the incoming packets.

As shown in figure 7, the filtering function uses a memory manager 560 having memory area (560M), and an incoming packet manager 550, comprising a set of associated (or "filtering") functions.

In a presently preferred embodiment, the memory area 560M is "reserved" statically for the filtering functions by the central processing unit (not shown) of the node; however, alternatively, the memory area 560M might be allocated dynamically as well, e.g. where the time needed for memory allocation is not penalizing.

The memory manager 560 may divide its memory area into portions of memory, which may be "allocated" and "released" dynamically, and individually. These portions of memory are used to store the above mentioned distinctive identifiers or "footprints".

The "filtering" functions 550 will be identified for convenience as follows:

- at 551, *search()* is a function which searches for a footprint in the memory area of 560, or in a part of it;
- at 552, *write()* is a function which writes a footprint in the memory area;
- at 553, *erase()* is a function which releases a portion of the memory area;
- at 555, *forward()* is a function sending a packet to the upper layers;
- at 556, *delete()* is a function deleting or throwing a packet;
- additionally, if it is desired to process message fragments, a *reassemble()* function at 559 gathers and reorders the fragments before they are forwarded to the upper layers, when the message is complete.

It will be noted that at least functions 551 through 553 interact with memory area 560.

Figure 7 also shows an independent process 569, named for convenience "IP_slowtimo", whose purpose is to release some part of memory area 560 from time to time.

In fact, the invention may be implemented by using software code, in which memory area 560 is represented by a memory manager (also denoted 560), capable of cooperating with memory hardware existing in the node, to have a reserved memory area, in which it defines at least one currently reserved portion of memory. Additionally, incoming packet manager 550 contains at least some of the filtering functions 551-559, depending upon the desired implementation.

The filtering method will be now described with reference to figures 8 and 9. Figures 8 and 9 show an exemplary detailed embodiment, comprising many features which are optional, i.e. not necessary in a simpler embodiment of this invention.

At 610, an IP packet (Pa) reaches protocol stack 10 of its final destination node Nj.

Frame 620 in figure 8 contains a set of optional operations, which are of interest when (1) high reliability and/or (2) cluster operation are desired.

Operation 621 checks whether the destination IP address (*IP-dest*) of the packet Pa matches the IP address (*IP-mlink*) of the multiple data link interface 101 of node Nj. If not, at 622, no filtering is made; for example, the packet may be subject to a "normal processing", through conventional IP functions 102. Other alternatives of operation 622, may be contemplated, e.g. considering a packet whose address does not match *IP-mlink* is in error. Operations 621 and 622 may even be omitted, if it is not desired to check the destination address.

Operation 625 checks whether the source IP address (*IP-src*) of packet Pa matches one of the cluster's node IP addresses (*IP-orig*) as stored in node Nj. In the example, *IP-orig* is a list containing the addresses (e.g. "intra-cluster" addresses) of all the nodes in the cluster. If desired, the list may exclude the local node. The list may also be restricted to those of the nodes in the cluster which are currently in operation.

If not, at 626, no filtering is made; for example, the packet may be subject to a "normal processing", through conventional IP functions 102. Other alternatives of operation 625 may be contemplated, e.g. considering a packet whose address does not match *IP-orig* is in error.

Operations 625 and 626 may even be omitted, if it is not desired to restrict filtering to intra-cluster messages.

The filtering per se begins at 630.

At 630, a value X is computed, using a function F(), such that a first incoming packet Pa1 and its redundant packet(s) Pa2 shall have the same value of X. Although it generally qualifies as a distinctive packet identifier, this value X is called a footprint hereinafter, for simplification.

Generally, when a packet is identified as a first incoming packet Pa1, i.e. footprint X is not found in memory area 560 at 632 (using the *search()* function), operation 640 stores the footprint X in a portion of memory area 560. A "Pa1" packet processing is made at 650. Otherwise, if X is found, a "Pa2" packet processing is made at 670. Finally, "end" operation 699 is reached.

Function F() is chosen, in combination with the internal structure of memory area 560, such that the risk that two packets Pa being not redundant with each other have the same footprint X is made as low as desired.

Generally (figure 10), memory area 560 may be organized as one or more tables T, in turn comprising sub-tables ST1 through ST_L. Parameter L enables to define an order between the sub-tables, or, at least, a currently in-use sub-table, e.g. ST1; preferably, an older one of the sub-tables, e.g. ST_L, is also defined, so as to authorize erasure of old footprints, as it will be seen hereinafter.

The terminology "tables" and/or "sub-tables" is used for clarity in the description; it must be understood that the tables and sub-tables are portions of memory, allocated by memory manager 560 within the memory area reserved to it.

In a first embodiment of this invention, memory area 560 is subdivided into a plurality of tables, each of which may be associated with a respective source address of the packets.

More precisely, a table Ti is dedicated dynamically to a source node Ni (in fact, except Nj), with i being an integer varying from 1 to C. C is the number of nodes in the cluster, or, more generally, the number of source addresses to be considered. A given source node may have several tables Ti. In the example, each table Ti is subdivided into sub-tables ST1-i through ST_L-i. The suffix "-i" is implicit in figure 10 and the sub-tables may be simply designated as ST1 through ST_L hereinafter.

In a specific embodiment (whether there is a single table T or several tables Ti), the sub-tables STm (with m from 1 to L)may be associated with pointers P[n] (with n also from 1 to L), such that each pointer P[n] designates a respective one of sub-tables STm. This will be explained in greater detail hereinafter. A given value of the pointer, e.g. P[1], designates the currently in-use sub-table, e.g. ST_L (Figure 10).

In the first embodiment, the X value as calculated by function F() need only to identify each IP packet among packets initiated by source node Ni to destination node Nj. Accordingly, the information from the IP header being used to determine the "footprint" does not include the source address.

Thus, for example when the packet is a data message, the value X may be derived from two fields of the IP header: field 226 or *IP-ID*, and field 224 or *IP-PROT*, e.g. using a concatenation of such fields. In a more precisely detailed embodiment, function F() may be a hash function of *IP-ID* and *IP-PROT* (or of *IP-ID* only, if a single protocol is being used).

In this case, the *search()* function of operation 632 is applied with footprint X to determine if X already exists in an allocated portion of the memory area, corresponding to node address *IP-orig* of node Ni, i.e. only in table Ti (if it already exists).

Turning to figure 9, operation 640 of figure 8 may be implemented in detail as follows:
- operation 641 looks whether a table Ti exists for the source node Ni, known from address *IP-orig*; if not, a table Ti is built at 643, e.g. by memory manager 560.
- operation 644 checks whether enough space is available for writing X in the current sub-table ST1 of Ti; if not, ST1 is expanded at 645; this may be viewed as allocates a new "current" portion of memory for node address *IP-orig*.

- then X may be written in the current sub-table ST1 of Ti at 648, and "end" 649 is reached.

In an embodiment, a given sub-table STm is adapted to store "footprints" of packets coming during a defined period of time. The term "footprint" designates an unique and single identification information about a packet, as described hereinabove. To avoid re-writing sub-tables STm, their chronological order may be defined by the pointers P[n].

In accordance with another aspect of this invention, the footprints are stored in the memory area for a limited time, which is called a critical age CA. When a footprint reaches the critical age, it may be deleted. The critical age CA may be a multiple of integer L. Thus, the erasure of footprints in tables Ti may be based on this discrete period of time CA/L.

In an embodiment, this erasure may be performed by independent process 569 (figure 7), named e.g. *IP-slowtimo*. This process may be called periodically by the system (or memory manager 560) to compute a clock, whose tick corresponds to CA/L. Assuming the process has been called at instant t1 for the first time, then a succession of clock instants t_{clock} may be defined, as shown by equation a. in Exhibit A, beginning with $t_{clock} = t1$.

At each such instant t_{clock}, the oldest sub-table or sub-tables STm, as designated by one of the pointer values P[n], e.g. P1, is or are cleared, i.e. released for receiving new data. The way the pointers P[n] represent the chronological order of sub-tables STm may have different forms. A simple form comprises using a cyclic permutation, such that, at each new clock instant t_{clock},:
- just before that instant, P[1] aims at the currently active sub-tables, while P[L] aims at the oldest sub-tables;
- at that instant, P[L] is erased, i.e. released for writing;
- immediately after that, P[1] becomes P[2], P[2] becomes P[3], ... P[L-1] becomes P[L], and P[L] becomes P[1], in accordance with the cyclic permutation, so that the new P[1] (formerly P[L]) becomes the currently active memory portion. The new P[L] (formerly P[L-1]), now aims at the oldest stored footprints.

Formally (in this embodiment):
- if a packet is received within the time interval defined by equation I1 in Exhibit A, where t represents any instant, and its footprint X is not yet stored in the table Ti, this footprint X is

stored in the cleared sub-table STm pointed during this time interval by the pointer P[1]. Each sub-table STm is only writable during the time interval it is pointed with the pointer P[1].

- more generally, at an instant t, sub-tables STm being aimed at by pointers P[n'] designate sub-tables having received packets within the time interval defined by equation I2 in Exhibit A, n' being an integer varying from 2 to L.

Now, the detailed implementation of operations 650 and 670 of figure 8 may depend upon whether the message in the current packet Pa is complete, or, by contrast, is a fragment of a message being split over several packets. In the example, IP header of a packet Pa, field 225 or "*IP-OFF*" contains indications of the existence of fragments, and of the position of the current fragment in the complete message, in the form of fragmentation flags, and fragment offsets. Operations 650 and 670 will be described in a version taking fragmentation into account; however, the operations directed to fragment processing may be omitted in certain cases, e.g. depending upon the target system, and the expected level of performance.

Figure 11 now shows an exemplary implementation of operation 650 of figure 8. Operation 652 determines whether a first incoming packet having footprint X is a fragment, e.g. from field 225. If not, the packet may be forwarded (function 555) to upper layers at 653.

The case of a fragmented message will now be considered.

A fragment queue (FQ(X)) designates a queue of fragments, all having the footprint X, in the course of being assembled to obtain a complete data message with the same footprint X. In fact, the *re-assemble()* function may be used to receive packets having the same X, and to re-order them into a data message, in accordance with the fragment offset data in the field *IP-OFF* of the IP header of each "fragment-containing" packet. This storage task may be devoted to memory manager 560, using its memory area (or a separate memory area), or to a separate process, again using the memory area of 560 or a separate memory area, and interconnected with multi data link interface 101.

It should also be kept in mind that the fragments do not necessarily arrive in their native order.

Reverting to figure 11, operation 654 determines whether there is already a fragment queue for X (and Ti, i.e. the source address *IP-orig*). If not, a fragment is built at 655. At 656, the incoming fragment is stored at 656, either at a location defined from its field 225 or "*IP-OFF*", or together with a representation of its location in the complete message, for later re-ordering, then going to "end" 669.

Figure 11 takes into account the fact that, being a first incoming packet with X, fragment Pa1 cannot already exist in the fragment queue.

Now, Figure 12 shows an exemplary implementation of operation 670 of figure 8 for a packet Pa2, i.e. a packet which is not the first incoming one with footprint X.

Operation 672 determines whether a packet Pa2 having footprint X is a fragment, e.g. from field 225. If not, the packet may be thrown or deleted (function 556) at 673, and "end" 699 is reached.

The case when Pa2 is a fragment shall now be considered.

Normally, a fragment queue already exists for X and Ti. However, if high reliability is desired, the existence of a fragment queue may be checked at 675 (the dashed line box illustrates the optional character of this). The operations in 675 may be similar to 654 and 655 in figure 11.

Then, operation 680 checks whether Pa2 is already present in fragment queue FQ(X). In the fragments have been immediately ordered, this may be made by determining whether the content at the fragment offset of Pa2 is occupied (and is Pa2, if desired); if the fragment offsets are stored in FQ(X) with the packets, this may be made by looking for the fragment offset of Pa2 amongst the fragment offset having already been stored.

If Pa2 is already present in fragment queue FQ(X), it may be thrown at 683, and "end" 699 is reached.

Otherwise, packet Pa2 is added to fragment queue FQ(X), at its location and/or together with a representation of its location, as previously.

Now, operation 686 checks whether the fragment queue is complete, again using the current and/or stored fragmentation data. If not, "end" 699 is reached. Otherwise, operation 687 may forward the complete message contained in FQ(X) to the upper layers. At 688, the memory for FQ(X) may be immediately released. Alternatively, it may be released at a later stage. It may be found interesting to release FQ(X) at the same time as X is released by the above mentioned *IP-slowtimo* process. Then, "end" 699 is reached.

If desired, the "fragment processing" operations in figures 11 and 12 may be merged into a single process, which is called when a fragment is detected in either case.

In an embodiment, the function F() may comprise a so-called hash function H, which spreads packets as uniformly as possible in a table Ti, e.g. T1-1. Such a hash function indicates at any instant if the sub-table STm pointed with the pointer P[1], is full or not. If the corresponding sub-table STm is full, the table T1-1 is considered to be full and the function H has to allocate a new table, e.g. T1-2, to store packet footprints in a sub-table STm designated with the pointer P[1] during a given time interval CA/L. This new table is allocated between certain not allocated reserved tables in memory area.

In other words, this new table T1-2 provides an additional storage space to filter packets coming from the same source node as table T1-1. When the function H determines that tables T1-1 and T1-2 have full sub-tables STm, an other new table may be added, e.g. T1-3. Each of these tables store footprints of packets coming from the same given source node, in the example.

Another optional task is to release tables without footprint in any of their sub-tables. In an embodiment, the process called *IP-slowtimo* further has to detect, at each CA/L, allocated tables whose each sub-table has no footprint. These tables are then released and considered not to be allocated any more, i.e. freed to be allocated again.

The above described method and system elements enable a fast discrimination and filtering of redundant packets, while taking into account fragmented messages, if desired. This is due inter alia to the dynamic allocation of memory portions within the whole memory area being allocated to memory manager 560.

As above described together with its alternatives, the first embodiment of this invention has advantages where messages are exchanged frequently between a restricted number of stations, e.g. within the nodes of a cluster, in fields like communications, business, government, education, entertainment, for example.

However, this invention is not limited to the hereinabove described features.

In a second embodiment, packets coming from several source addresses might be stored commonly in the same portion of memory, i.e. in a single table T. If so, the footprint function F() should be supplemented to take the source address into consideration. In this case, the *search()* function of operation 632 in figure 8 would explore all allocated portions of the memory area, or even the whole memory area.

In this second embodiment, operations 641 and 642 of figure 9 may be changed, or even omitted, to the extent one or more packets coming from another source address have already caused a table T (more broadly, a memory portion) to be allocated, and written with the X values of these previous packets. In fact, if all incoming packets are stored in common in the whole memory area, there may be no need to implement operations 641 and 642.

In an alternative of the second embodiment, it may also be contemplated that tables Ti are dedicated to groups of source nodes, instead of individual source nodes. If so, the footprint function F() should be supplemented to take the source address into consideration, at least partially.

A third embodiment of this invention is an alternative for all the preceding ones. It concerns the processing of fragment packets. In this case, the value X takes into account the fragment identifying data, at least in part. For example, the value X may be defined from a concatenation of the value *IP-ID*, *IP-PROT*, and *IP-OFF* of the IP header (and also *IP-orig*, if a single table T is being used). In this case, each fragment carrying packet is processed like the packets having complete messages. In other words:

- in the firstly described embodiment, only the first incoming fragment packet is considered as Pa1; all subsequent packets are treated as Pa2, independently of their fragment offset, and of the fact they are first arrived, or redundant, for their particular offset;

- in the third embodiment, an incoming packet is considered as Pa1 if it is the first arriving for its fragment offset; thus, a packet seen as Pa2 is redundant with a previously received packet having the same offset.

In an version of that embodiment, operation 640 in figure 8 may be integrated to the Pa1 packet processing in operation 650.

In a fourth embodiment of this invention, where a fragment carrying packet is concerned, no footprint X (of fragment) is written in memory 560 at operation 640 (figure 8). Thus, only footprints of complete message packets are written in memory 560. The fragment carrying packets are not be filtered by using the footprint. Instead, all fragments are stored in the fragment queue, whether redundant or not. When the message is complete (i.e. all "offsets" are present, whether redundant or not), the *reassemble()* function gathers the whole message and ignores the redundant fragments. In other words, the filtering of fragment packets is made in the fragment queue, at the time the *reassemble()* function is executed.

The reassemble() function identifies the identical duplicated fragments with the value X being a concatenation of the value *IP-ID*, *IP-PROT*, and *IP-OFF* of the IP header. The reassemble() function throws duplicated fragments. Once a fragmented data message is reassembled, the fragment queue is freed and the complete data message is forwarded to upper layer.
Those skilled in the art will note that any subsequently arriving fragment packets, being redundant with the just completed message, will create a new fragment queue, which is not useful.

In an alternative of the fourth embodiment, where fragment carrying packets are concerned, the footprint X is written in memory 560 only after the complete message has been re-assembled. In this case, the footprint X need not comprise any fragment identification data. Thus, both footprints of complete message packets and of reassembled fragmented data messages are written in memory 560. These operations may be added in figure 11, after operation 656. Thus, any subsequently arriving fragment packets are filtered. Indeed, after forwarding a reassembled fragmented data message and storing its footprint in memory 560, the duplicated fragment carrying packets corresponding to this footprint in operation 632, figure 8, are thrown, in

operation 670. Thus, no duplicated fragment of a fragmented data message is stored in a new fragment queue.

This invention also covers the software code for performing the method, especially when made available on any appropriate computer-readable medium. The expression "computer-readable medium" includes a storage medium such as magnetic or optic, as well as a transmission medium such as a digital or analog signal. The software code basically includes separately or together, the codes defining the memory manager 560, the packet manager 550, and the codes for implementing at least partially the flow-charts of figures 5, 6,8 ,9 11 and 12.

Exhibit A

a.

$t_{clock} = t1+pxCA/L$

I1.

[t-(t-t1)MOD(CA/L), t]

I2.

[t-(t-t1)MOD(CA/L)-(n'-1)xCA/L,t-(t-t1)MOD(CA/L)-(n'-2)xCA/L]

Claims

1. The software code, comprising code for defining:
- a memory manager (560), having a reserved memory area, in which it defines a currently designated portion of memory, and
- an incoming packet manager (550), responsive to an incoming packet for:
    * determining an identifier (X) from information contained in the incoming packet,
    * searching the identifier (X) in at least a portion of the reserved memory area,
    * if a first condition related to the search is met, storing the identifier (X) in the currently designated portion of memory, and
    * if a second condition related to the search is met, filtering out the incoming packet as redundant.

2. The software code of claim 1, wherein:
- the first condition comprises the identifier (X) being not found, and
- the second condition comprises the identifier (X) being found.

3. The software code of claim 1 or claim 2, wherein:
- the memory manager (560) is capable of defining a fragment memory area (FQ(X)) in its reserved memory area, and
- responsive to a third condition, comprising the fact an incoming packet contains a message fragment, the incoming packet manager (550) stores at least a portion of the packet in the fragment memory area (FQ(X)).

4. The software code of claim 3, further comprising a mechanism for re-assembling a message when all its fragments have been stored in the fragment memory area (FQ(X)).

5. The software code of claim 3, wherein:
- the identifier (X) is determined from information which includes fragment identifier information, and
- the third condition comprises the incoming packet containing a message fragment and its identifier being not found.

6. The software code of claim 3, wherein:
- the identifier (X) is determined from information other than fragment identifier information,
- the first condition comprises the identifier (X) being not found, and
- the second condition comprises the identifier (X) being found and the incoming packet data containing an unfragmented message.

7. The software code of claim 6, wherein:
- the first condition comprises the identifier (X) being not found and the incoming packet containing an unfragmented message.

8. The software code as claimed in any of the preceding claims, wherein:
- the memory manager (560) is capable of defining selected sub-areas (Ti; T1-1) in the reserved memory area, and of defining a selected currently designated portion of memory in each such selected sub-area,
- the identifier (X) is determined from information in the packet other than source-related information,
- the incoming packet manager (550) performs the search is a selected sub-area associated to the source address of the incoming packet, if any, and
- if the first condition related to the search is met, the identifier (X) is stored in a currently designated portion of memory of a selected sub-area associated to the source address of the incoming packet.

9. The software code as claimed in any of the preceding claims, further comprising a monitoring timer mechanism (569), capable of releasing areas in the reserved memory which contain older identifiers (X).

10. The software code of claim 9, wherein:
- the memory manager (560) keeps a chronological track of prior designated portions of memory, and
- the monitoring timer (569) periodically orders the memory manager (560) to release oldest ones of the designated portions of memory.

11. The software code as claimed in any of the preceding claims, wherein the memory manager (560) comprises a hash function for storing data in the reserved memory area.

12. The software code as claimed in any of the preceding claims, wherein the memory manager (560) is arranged to assign an additional currently designated portion of memory (T1-2), when a currently designated portion of memory (T1-1) is full.

13. The software code as claimed in any of the preceding claims, wherein the first and second conditions each comprise the destination address of the incoming packet matching a proper address (IP_10).

14. A method of processing redundant packets, comprising the steps of:

a. determining an identifier (X) from information contained in an incoming packet (630),

b. searching the identifier (X)in at least a portion of a reserved memory area (560), and

c. processing the packet, the processing comprising:

c1. if a first condition related to the search is met, storing the identifier (X) in a currently designated portion of the reserved memory area, and

c2. if a second condition related to the search is met, filtering out the incoming packet as redundant.

15. The method of claim 14, wherein:

- the first condition comprises the identifier (X) being not found, and
- the second condition comprises the identifier (X) being found.

16. The method of claim 14 or claim 15, wherein step c. further comprises:

c3. responsive to a third condition, comprising the fact an incoming packet contains a message fragment, storing at least a portion of the packet in a fragment memory area (FQ(X)).

17. The method of claim 16, wherein step c. further comprises:

c4.. re-assembling a message when all its fragments have been stored in the fragment memory area (FQ(X)).

18. The method of claim 16, wherein:

- step a. comprises determining the identifier (X) from information which includes fragment identifier information, and

- the third condition of step c3. comprises the incoming packet containing a message fragment and its identifier (X) being not found.

19. The method of claim 16, wherein:

- step a. comprises determining the identifier (X) from information other than fragment identifier information,

- the first condition of step c1. comprises the identifier (X) being not found at step b., and

- the second condition of step c2. comprises the identifier (X) being found at step b. and the incoming packet data containing an unfragmented message.

20. The method of claim 19, wherein:

- the first condition of step c1. comprises the identifier (X) being not found at step b. and the incoming packet containing an unfragmented message.

21. The method as claimed in any of claims 14 through 20, wherein:

- step a. comprises determining the identifier (X) is determined from information other than source-related information,

- step b. comprises searching the identifier (X) in a selected sub-portion, if any, which is associated to the source address of the incoming packet, and

- step c1. comprises: if the first condition related to the search is met, storing the identifier (X) in a currently designated sub-portion of the reserved memory area, which is associated to the source address of the incoming packet.

22. The method as claimed in any of claims 14 through 21, further comprising the step of:

d. at time intervals, releasing areas in the reserved memory which contain older identifiers (X).

23. The method of claim 22, wherein step d. comprises:

d1. keeping a chronological track of prior designated portions of memory, and

d2. periodically releasing oldest ones of the designated portions of memory.

24. The method as claimed in any of claims 14 through 23, wherein step c. comprises using a hash function for storing data in the reserved memory area.

25. The method as claimed in any of claims 14 through M24, wherein:
- step c1. comprises using an additional currently designated portion of memory (T1-2), when a currently designated portion of memory (T1-1) is full.

26. The method as claimed in any of claims 14 through 25, wherein the first and second conditions each comprise the destination address of the incoming packet matching a proper address (IP_10).

27. In a node having a protocol stack (10), a multiple data link interface (110), capable of at least partially implementing the functions in the method of any of claims 14 through 26.

28. A cluster having a plurality of nodes interconnected through a redundant link, in which at least two of the nodes are arranged for implementing the functions in the method of any of claims 14 through 26.

FIG.1

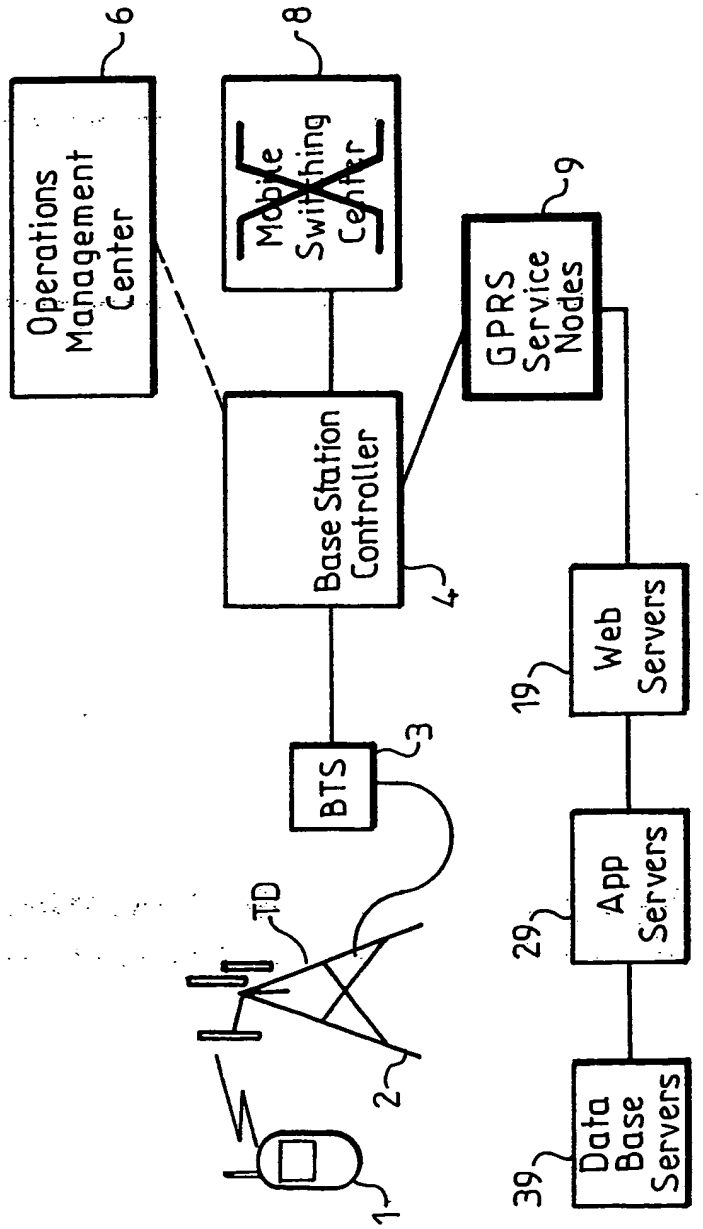
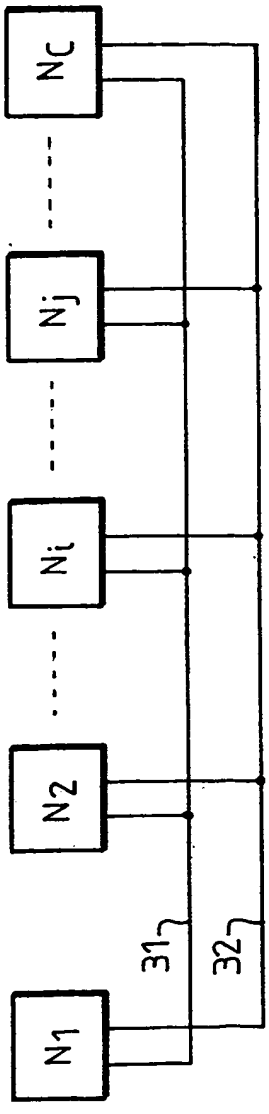


FIG.2



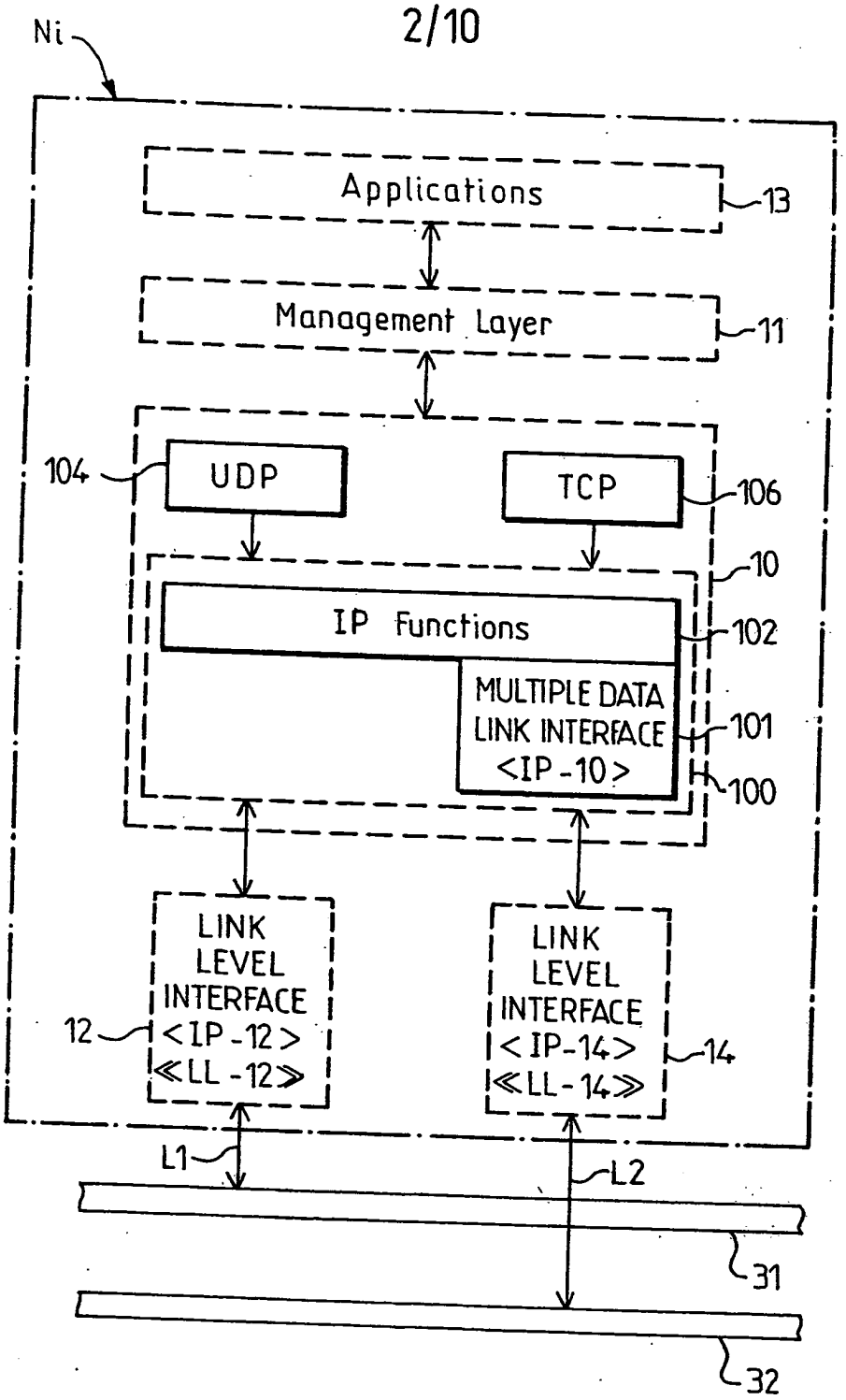


FIG.3

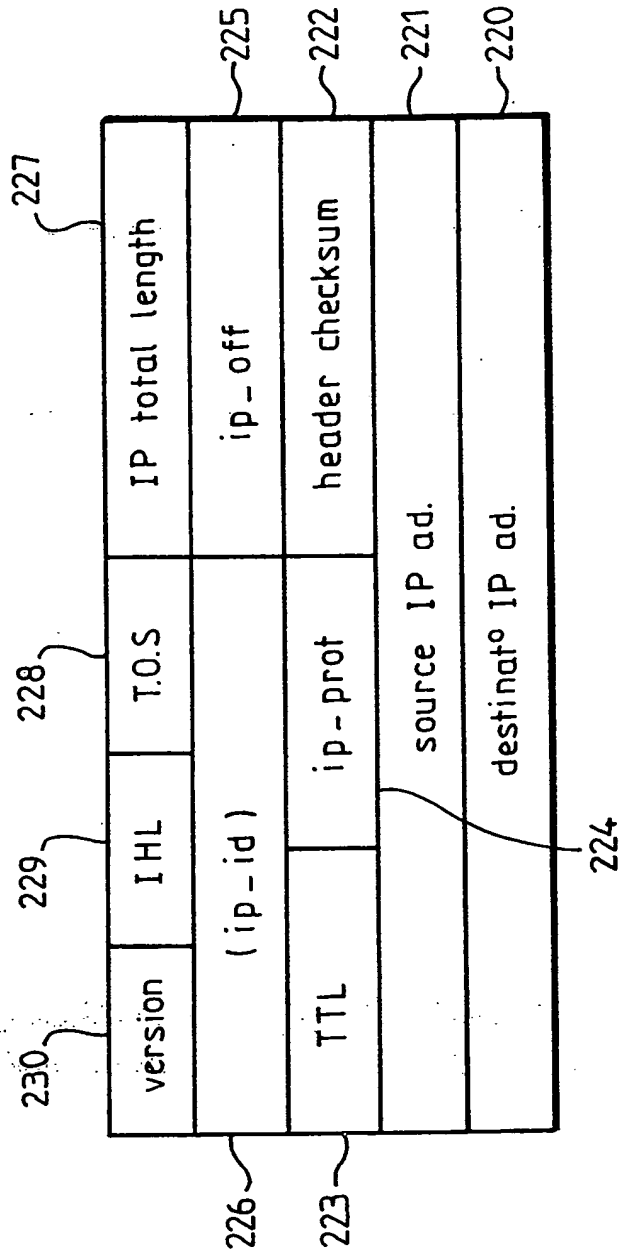


FIG.4

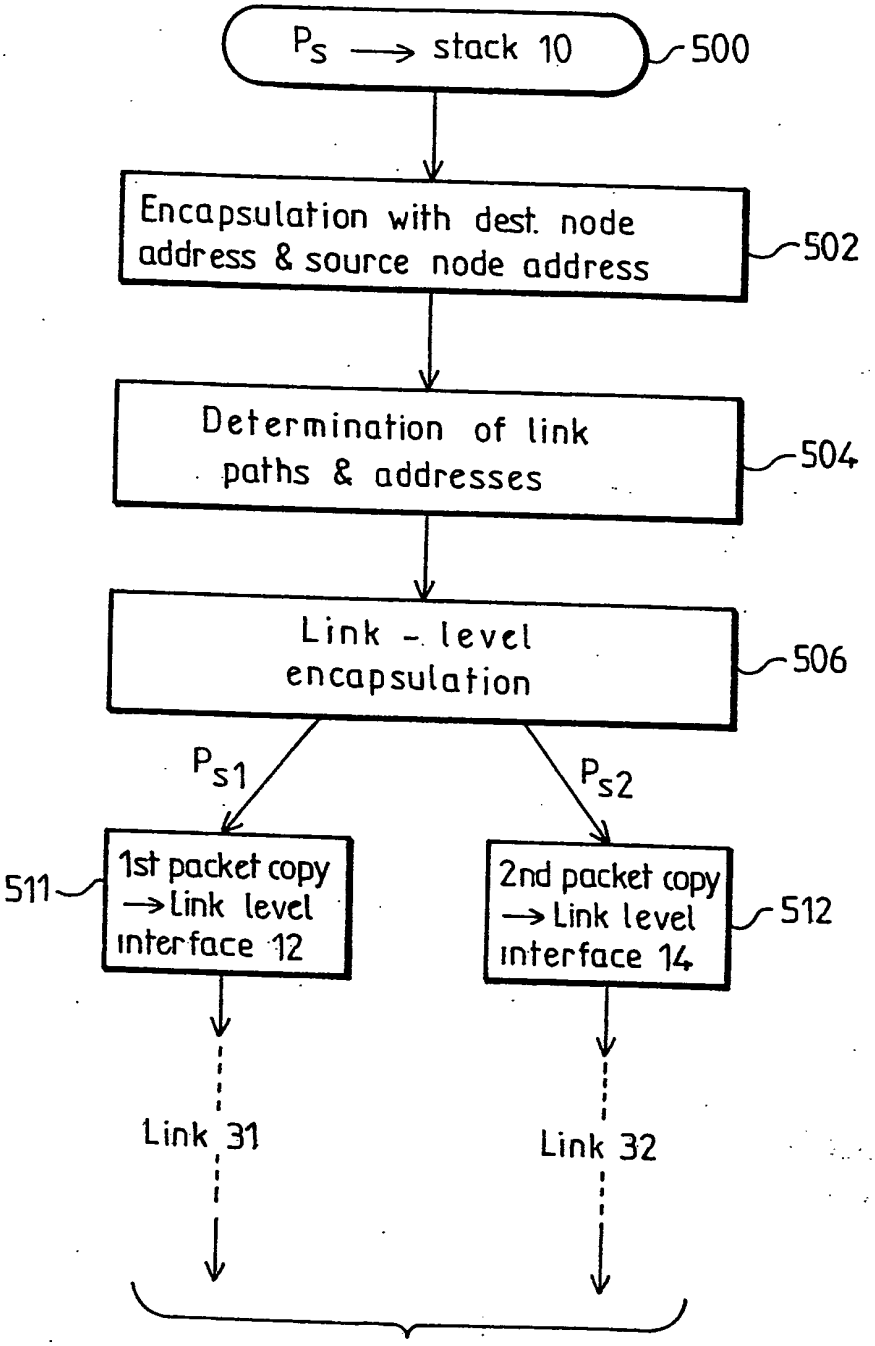


FIG.5

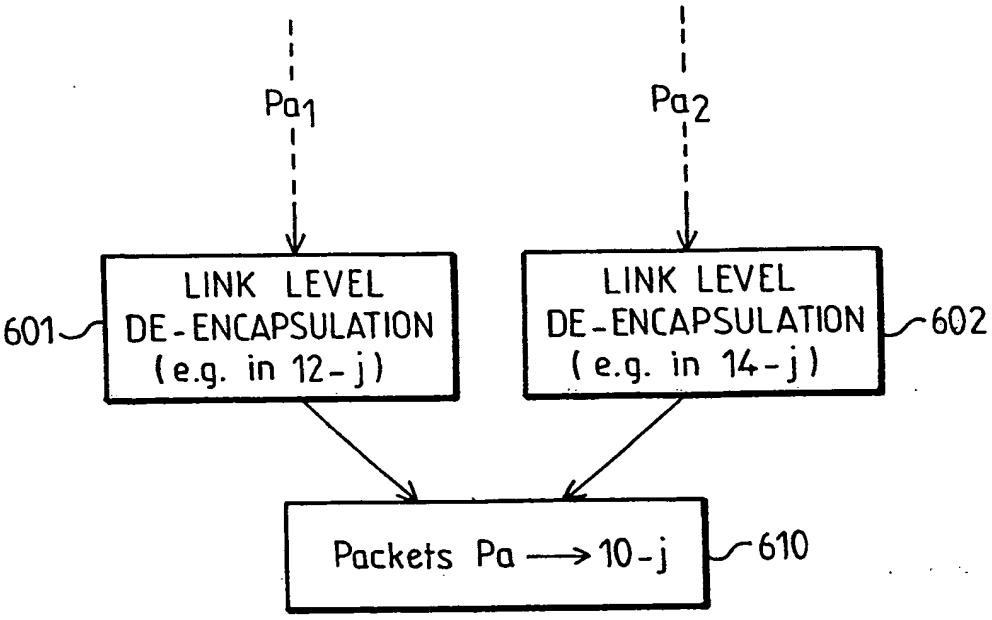


FIG.6

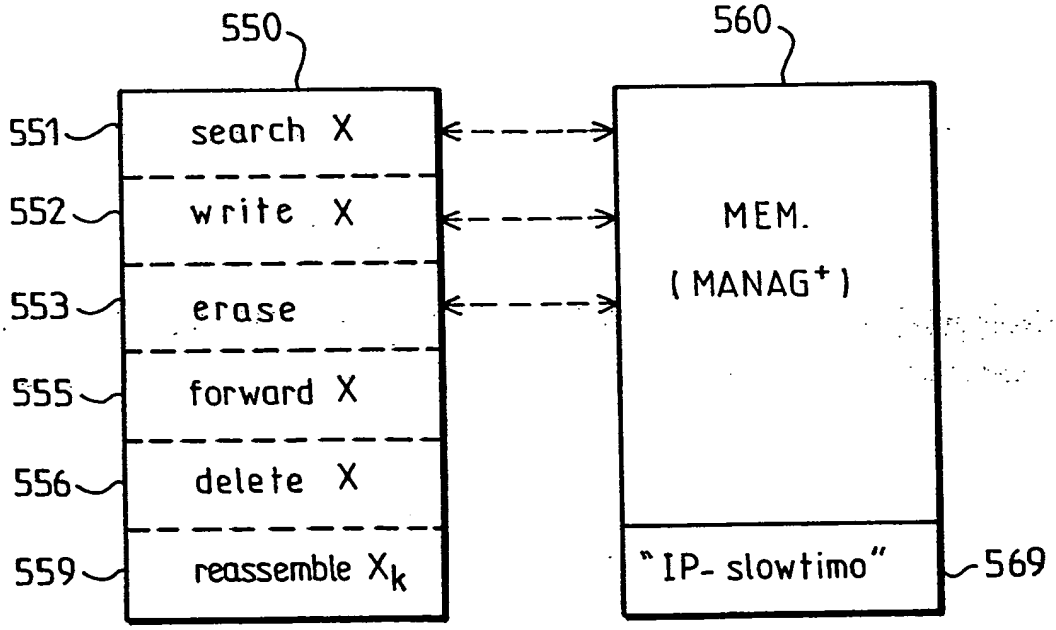


FIG.7

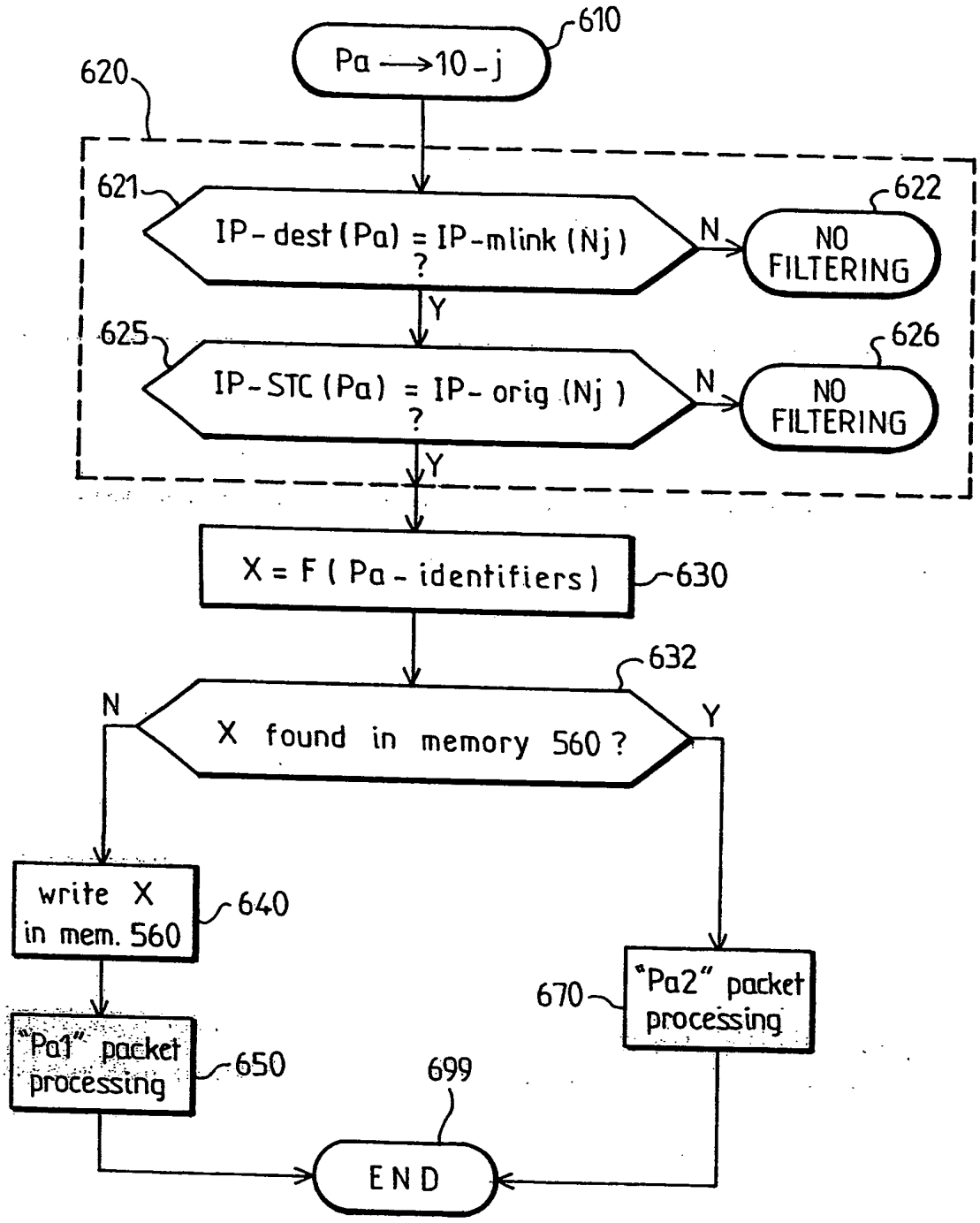


FIG.8

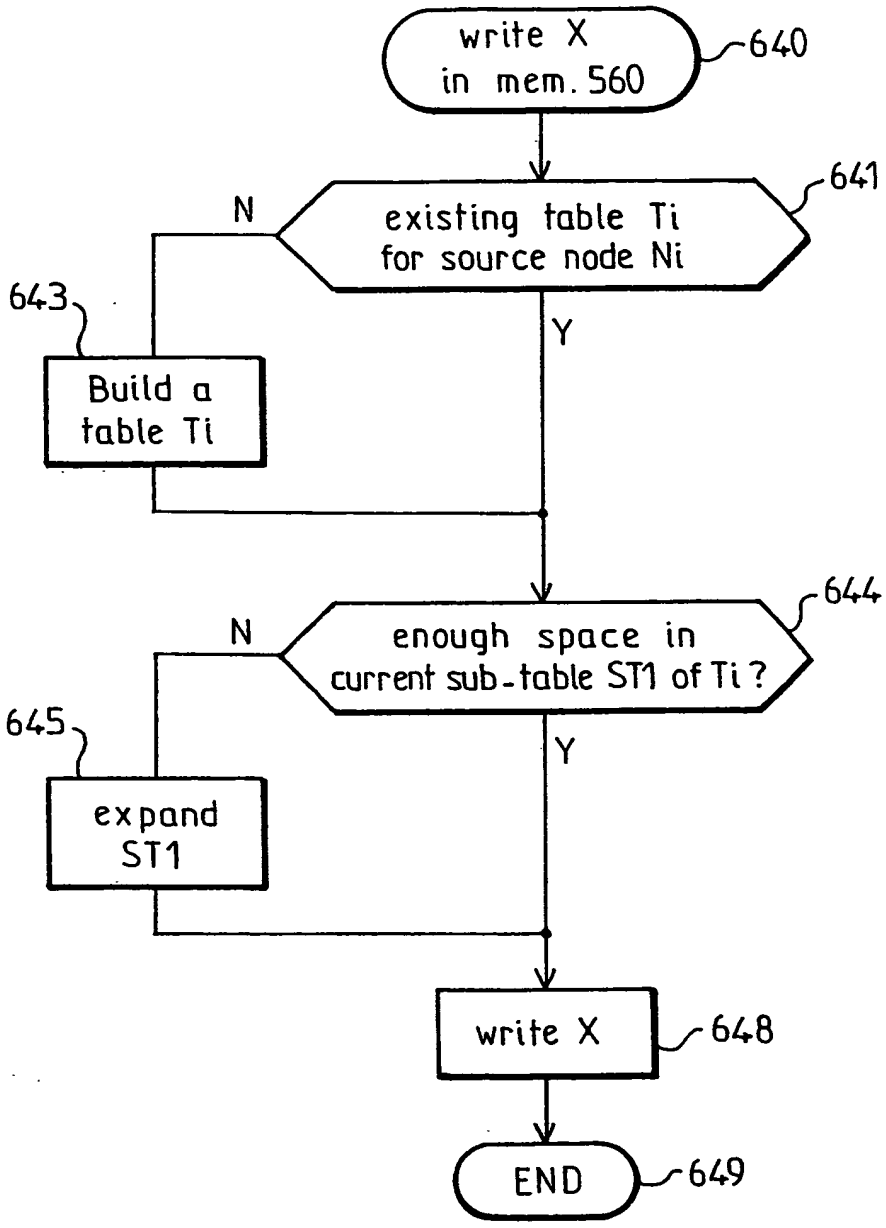


FIG.9

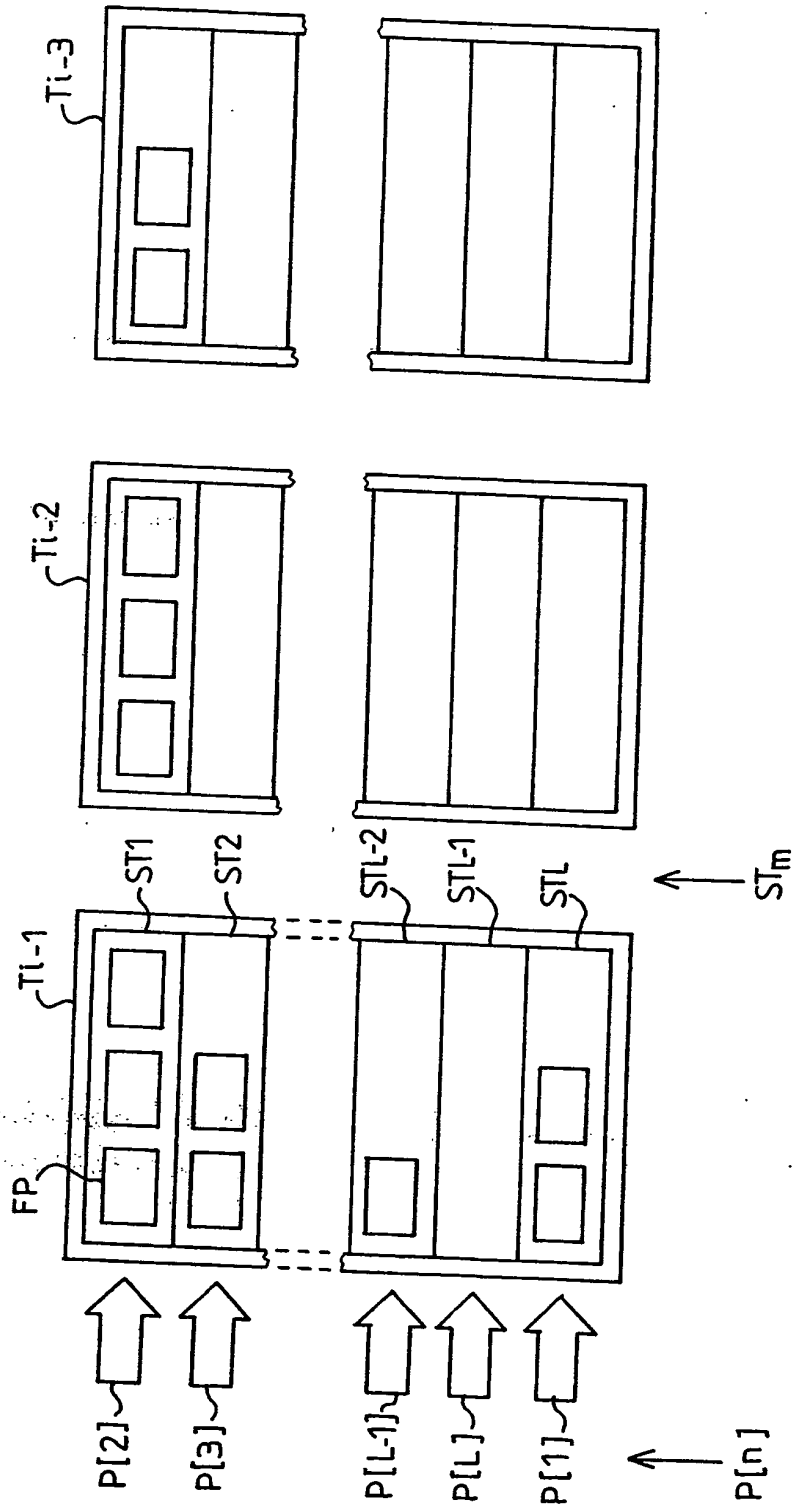


FIG.10

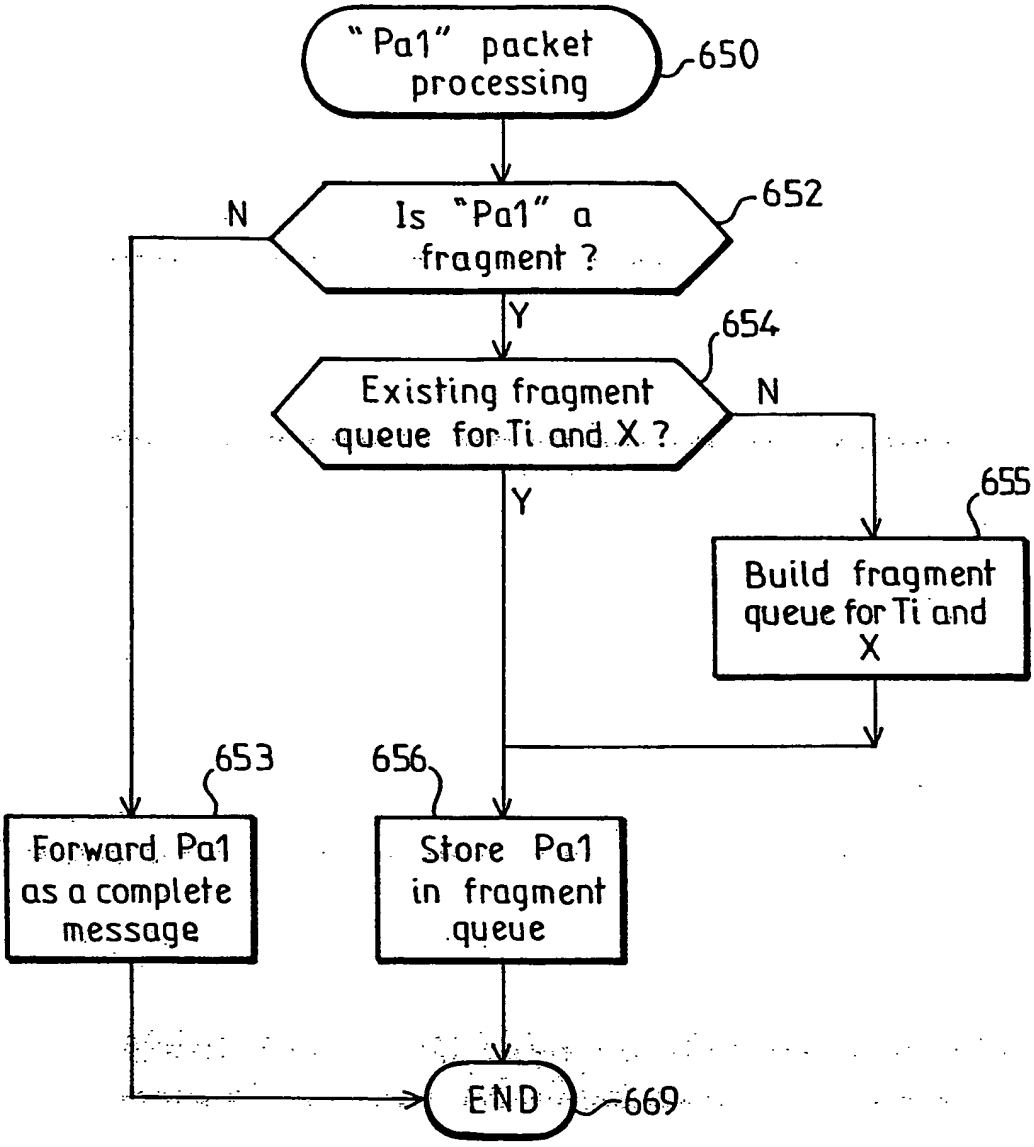


FIG.11

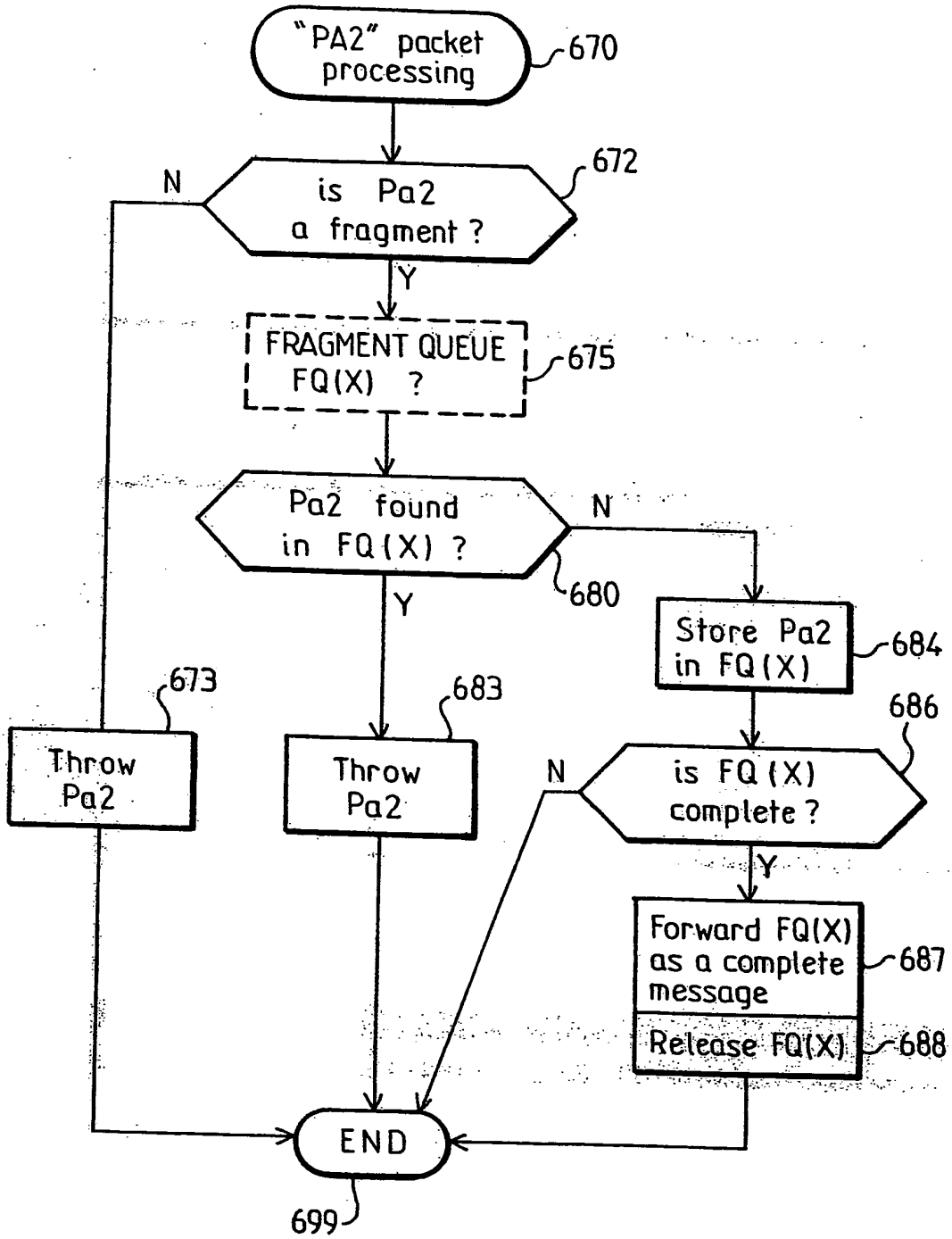


FIG.12

INTERNATIONAL SEARCH REPORT

Int onal Application No
PCT/IB 01/01382

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 H04L29/06 H04L29/14

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 H04L

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal, WPI Data, PAJ

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	WO 00 56024 A (BROADCOM CORP ;KADAMBI SHIRI (US); AMBE SHEKHAR (US); JORDA MICHAE) 21 September 2000 (2000-09-21) page 2, line 5 -page 5, line 22 page 8, line 18 -page 20, line 7 page 25, line 4 -page 43, line 17 abstract; figures 1,5-9,11-14,17-22	1-5, 8-18, 21-28
A	---	6,7,19, 20
X	US 5 832 233 A (HALL TREVOR ET AL) 3 November 1998 (1998-11-03) the whole document	1-7, 14-20, 27,28
	-/--	

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
11 April 2002	19/04/2002

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL – 2280 HV Rijswijk Tel. (+31–70) 340–2040, Tx. 31 651 epo nl, Fax (+31–70) 340–3016	Kalabic, F

2

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Int onal Application No
PCT/IB 01/01382

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
A	WO 98 18247 A (GARCIA GOMEZ FRANCISCO MANUEL ;MERCHAN SANZANO RAMON (ES); PLAZA F) 30 April 1998 (1998-04-30) page 3, line 10 -page 13, line 20 page 21, line 1 -page 25, line 19	1-28

2

Form PCT/ISA/210 (continuation of second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No
PCT/IB 01/01382

Patent document cited in search report		Publication date	Patent family member(s)		Publication date
WO 0056024	A	21-09-2000	US	2001043611 A1	22-11-2001
			AU	3529500 A	04-10-2000
			AU	3752000 A	04-10-2000
			AU	4325800 A	04-10-2000
			EP	1161817 A2	12-12-2001
			EP	1169809 A2	09-01-2002
			EP	1159805 A2	05-12-2001
			WO	0056024 A2	21-09-2000
			WO	0056011 A2	21-09-2000
			WO	0056013 A2	21-09-2000
			AU	5442700 A	12-12-2000
			AU	5586400 A	12-12-2000
			AU	5737300 A	31-01-2001
			AU	6334400 A	22-01-2001
			EP	1181791 A1	27-02-2002
			EP	1181792 A1	27-02-2002
			EP	1192767 A2	03-04-2002
			WO	0072533 A1	30-11-2000
			WO	0102965 A2	11-01-2001
			WO	0072531 A1	30-11-2000
			WO	0101724 A2	04-01-2001
			US	2002031090 A1	14-03-2002
US 5832233	A	03-11-1998	GB	2304505 A ,B	19-03-1997
WO 9818247	A	30-04-1998	AU	5313498 A	15-05-1998
			WO	9818247 A1	30-04-1998
			EP	0944981 A1	29-09-1999